This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE

IEEE

Membership Publications/Services Standards Conferences Careers/Jobs

IEEE Xplore® RELEASE 1.6

Welcome
United States Patent and Trademark Office

Help FAQ Terms IEEE Peer Review Quick Links » Search Results

Welcome to IEEE Xplore®
- Home
- What Can I Access?
- Log-out

Tables of Contents
- Journals & Magazines
- Conference Proceedings
- Standards

Search
- By Author
- Basic
- Advanced

Member Services
- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

Your search matched **1** of **990941** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

decoding and instruction and predecoder Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Pre-decoding mechanism for superscalar architecture**
Minagawa, K.; Saito, M.; Aikawa, T.;
Communications, Computers and Signal Processing, 1991., IEEE Pacific Rim Conference on , 9-10 May 1991
Pages:21 - 24 vol.1

[Abstract] [PDF Full-Text (208 KB)] **IEEE CNF**

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | O
Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ | Terms | Back to

Copyright © 2004 IEEE — All rights reserved

h eee e eee g e ch e ch e e ec g c e ec e c

IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, May 9-10, 1991

PRE-DECODING MECHANISM FOR SUPERSCALAR ARCHITECTURE

Kenji Minagawa Mitsuo Saito Takeshi Aikawa

Toshiba Research and Development Center
1,komukaitoshibamachi, Saiwai-ku, Kawasaki, JAPAN

Abstract

This paper describes a pre-decoding mechanism for superscalar processors, which is necessary to issue instructions to plural functional units. Using a pre-decoding mechanism, a processor can issue instructions without increasing the cycle time or the branch penalty. The pre-decoder decides which functional unit should execute an instruction during a cache miss and refill. A special tag, which we call pre-decode tag, is generated by the pre-decoder. This tag is saved with instructions to an instruction cache memory. The processor fetches instructions with the pre-decode tag and issues instructions controlled by the pre-decode tag. Using this mechanism, fast cycle times, comparable to RISC processors, are attained.

1.Introduction

RISC processors already achieve an instruction-execution rate of nearly one instruction per cycle. To attain greater performance, several approaches to multiple instruction execution per cycle are used. A superscalar approach and a VLIW approach are typical ones. Superscalar processors take advantage of maintaining scalar code compatibility, whereas VLIW processors do not. However, superscalar processors need an operand dependency analysis mechanism and an instruction issuing mechanism.

Like RISC processors, superscalar processors should be designed with a short cycle time and a low branch penalty. The register scoreboarding technique is one approach to operand dependency analysis. The Motorola M88000 RISC processer[1] employs this technique(the register scoreboarding technique) to execute floating point operations having latencies. It analyzes the operand dependency and reads a register file in the decode stage. Using this register scoreboarding technique, the time required for operand dependency analysis is comparable to that of a register read access. Therefore the operand dependency analysis mechanism is not an obstacle to attaining short cycle times comparable to conventional RISC processors.

On the other hand, the instruction issuing process must occur before the register read process. Executing the instruction issuing process and register read access in one cycle becomes a problem when trying to attain the fast cycle times.

The IBM RS6000[2] has a dispatch pipeline stage for issuing instructions before the decode stage when it reads the register file. Such a pipeline stage strategy increases the branch penalty. To compensate this side effect, the RS6000 uses a zero time branch mechanism. However, to use this mechanism effectively, the compiler should move the condition setting code instructions far ahead of the conditional branch instructions. Therefore, the availability of this mechanism is limited.

Our pre-decoding mechanism is developed for issuing instructions without increasing the branch penalty or the cycle time.

2.Pre-decoding Mechanism Policy

Our policy is to issuing instructions and execute the register read access in the decode pipeline stage of one cycle and reduce the instruction issuing process in that stage. The instruction issuing process is divided into two process. One is deciding which functional units should execute instructions. The other is sending instructions via a buffer to appropriate functional units. We choose the former process to be executed in a cache miss and refill process and the later process to be executed in the decode pipeline stage. Although the instruction cache miss penalty increases, such an increase degrades the processor performance little.

3.Pre-decoder

Figure 1 shows a system using the pre-decoding mechanism. A pre-decoder decides which functional unit should execute an instruction during a cache miss and refill.

When a cache miss and refill occurs, instructions are transferred from main memory. The pre-decoder generates a pre-decode tag by decoding the instructions. The pre-decode tag indicates which functional units can execute instructions, avoiding functional unit conflicts†. The pre-decode tag and the instructions are saved to instruction cache memory. After a cache miss and refill is completed, the processor fetches the instructions with the pre-decode

† More than two instructions are issued to a same functional unit

CH2954-6/91-0000-21 $1.00 1991 IEEE

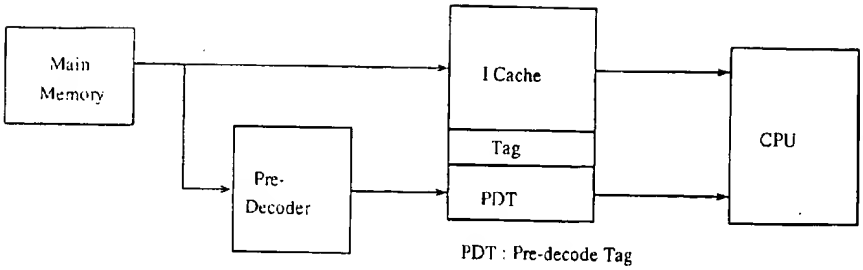


Figure 1. Pre-decode Tag System

Rn : Resource Field

Pn : Priority Field

Figure 2. Pre-decode Tag

tag.

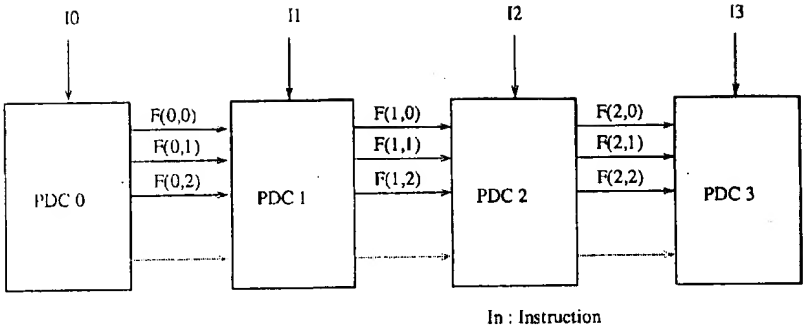
It is necessary to dynamically analyze the operand dependency when the processor executes instructions, such as floating point operations which have latency. The pre-decoder cannot analyze the operand dependency, therefore, the register scoreboard dynamically analyzes it.

Figure 2 shows the fields of the pre-decode tag. If the processor has N functional units, which can be executed simultaneously, the pre-decode tag will have N resource fields and (N-1) priority fields. The resource field indicates which functional units should execute instructions. The priority field indicates whether the instruction can be executed simultaneously with the previous instruction. If there aren't enough functional units which execute simultaneous instructions, the priority field will be asserted.

Figure 3 shows a block diagram of the pre-decoder. The PDC(pre-decode cell) i generates the priority field and the resource field of the instruction i. The PDC i sends the signals F(i,j) to the PDC i+1. Most superscalar processors have different type of functional units, for example integer units and floating point units, which can be executed simultaneously. The value j of the signals F(i,j) indicates the type of functional units which will execute the instruction i. The priority field of an instruction i is generated by the following algorithm:

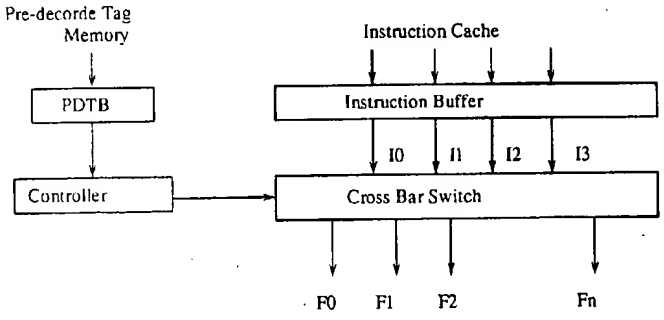
F(0,m) = 0.
M is the number of m-type functional units.
If an instruction i can be executed by an m-type functional unit, T = F(i-1,m) + 1.
If T > M, then F(i,m) = 1; the priority field will be asserted.
If T <= M, then F(i,m) = Tm; the priority field will be negated.
If an instruction i cannot be executed by an m-type functional unit, F(i,m) = F(i-1,m).

A resource field code is assigned to each functional unit. If the processor has just one functional unit of some type, the resource fields of instructions executed by this functional unit will simply be generated by decoding the instruction's operation code. If the processor has more than two functional units of some type, it is necessary to decide which functional unit of this type will execute the preceding instruction. Corresponding to this policy, the resource fields of instructions executed by such functional units are generated. For example, if the processor has two integer units A and B, it will be decided that A executes the preceding instruction and B executes following one. At first, the pre-decoder assigns an integer operation instruction to A. If A is already assigned an instruction, the pre-decoder will assign an integer operation instruction to B.



In : Instruction

Figure 3. Pre-Decoder



Fn : Functional Unit
PDTB : Pre-Decode Tag Buffer

Figure 4. Instruction Issuing Unit

4.Instruction Issuing Unit

Figure 5 shows an instruction issuing unit. Instructions in an instruction buffer are issued to functional units through a cross bar switch circuit, controlled by the pre-decode tag in a pre-decode tag buffer. The functional units, which receive issued instructions, decode the instructions and read register files. In addition, the instructions in the instruction buffer are analyzed for operand dependency by the register scoreboard. The instruction issuing, operand dependency analysis, decoding and register reading are done in one pipeline stage in oder to not increase the branch penalty.

If the priority fields of an instruction i and an instruction j (i<j) are asserted and the priority fields of all instructions k (i<k<j) are negated, the cross bar switch circuit will issue all instructions m (i<=m<j) simultaneously. If the issued instruction has dependency on a previous instruction, the register scoreboard will cancel the issuing of that instruction, sending a cancel signal to the functional unit where the instruction is issued. This instruction is canceled, will be issued again by the cross bar switch circuit in the next cycle. If the issuing of all instructions m (i<=m<j) is completed, then the cross bar switch will issue l (l >= j),

5.Example

	RS	PF
(1) add r0,r1,r2	0	
(2) add r2,r3,r4	1	0
(3) add r5,r6,r6	0	1
(4) fadd f0,f2,f3	2	0
(5) fmul f4,f5,f6	3	

RS: resource field PF: priority field

Assignment of resource fields

integer unit 1	0
integer unit 2	1
floating adder unit	2
floating multiplier unit	3

Figure 5. Example Code

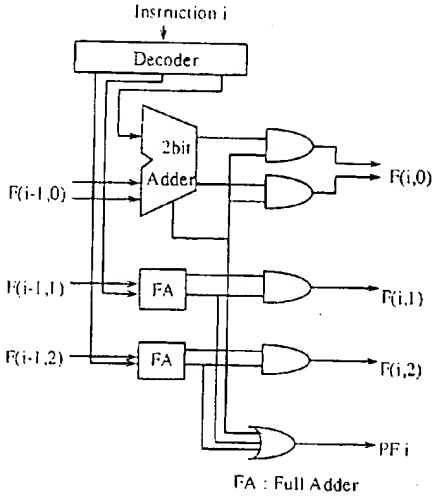


Figure 6. PDC

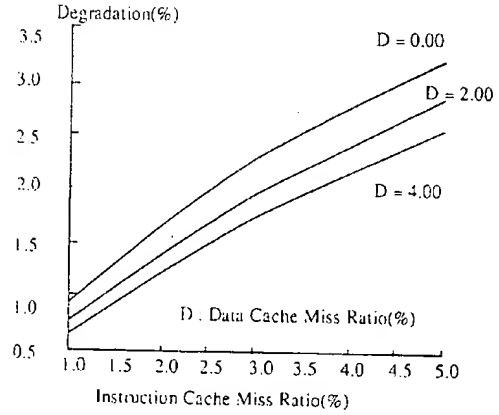


Figure 7. Perfomance Degradation

Figure 5 shows an example of assembler codes. We assume that the processor has two integer units, a floating adder unit and a floating multiplier unit and that it can execute a maximum of four instructions. When an instruction cache miss and refill occurs, the pre-decoder analyzes the instructions (1) through (4). The priority field of instruction(3) is asserted because the processor can only execute two integer operations simultaneously.

After the instruction cache miss and refill is completed, the processor fetches instructions (1) through (4) with the pre-decode tag. The processor only issues instructions (1) and (2) because the priority field of instruction(3) is asserted. After issuing instructions (1) and (2), the processor issues instructions (3) and (4). When all four instructions have been issued to functional units, the processor fetches the next four instructions.

6.Evaluation

Our pre-decode mechanism increases the cache miss and refill time. We evaluated this side effect by measuring the gate delay time for a pre-decoder of a processor containing two ALU, a floating multiply unit and a floating adder unit.

Figure 6 shows a circuit of a PDC. The pre-decoder can be constructed with four PDC. F(i,0) indicates the number of ALU operations. F(i,1) indicates the number of floating add operations. F(i,2) indicates the number of floating multiply operations. A critical path of this pre-decoder is instruction 0 to PF3. Using the 0.8 micron CMOS logic, the gate delay time of this pre-decoder is 9.8 nsec. This delay time is shorter than the CPU cycle time because it takes 10 nsec to analyze the operand dependency.

Figure 7 shows the performance degradation due to the pre-decoding mechanism, assuming that the cache refill time and the pre-decode time are 10 cycles and 1 cycle, respectively. If the instruction cache size is 16 Kbyte, the instruction cache miss ratio is 2% according to [3]. If the data cache size is 16 Kbyte, then the data cache miss ratio is 3% and the performance degradation is 1.3%. This degradation is much less than that of increasing the cycle time for issuing instructions or having extra stages in the decoder.

7.Conclusion

Using the pre-decode mechanism, the superscalar processors attain faster cycle times, comparable to RISC processors, without increasing the branch penalty. The side effect of this mechanism is an increase in the instruction cache miss and refill time. However, the degradation of the performance appears to be low.

References

[1] MC88100 User's Manual, MOTOLORA INC, 1988.
[2] IBM RISC System/6000 Technology, IBM Corporation, 1990.
[3] A.J.Smith, "Cache Memories," Computing Surveys, Vol. 14, No.3 September 1982, pp.473-530.